```
File   8:Ei Compendex(R) 1970-2004/Jul W2
         (c) 2004 Elsevier Eng.  Info. Inc.
File  35:Dissertation Abs Online 1861-2004/May
         (c) 2004 ProQuest Info&Learning
File 202:Info. Sci. & Tech. Abs. 1966-2004/Jul 12
         (c) 2004 EBSCO Publishing
File  65:Inside Conferences 1993-2004/Jul W3
         (c) 2004 BLDSC all rts. reserv.
File   2:INSPEC 1969-2004/Jul W2
         (c) 2004 Institution of Electrical Engineers
File  94:JICST-EPlus 1985-2004/Jun W4
         (c)2004 Japan Science and Tech Corp(JST)
File 483:Newspaper Abs Daily 1986-2004/Jul 19
         (c) 2004 ProQuest Info&Learning
File   6:NTIS 1964-2004/Jul W3
         (c) 2004 NTIS, Intl Cpyrght All Rights Res
File 144:Pascal 1973-2004/Jul W2
         (c) 2004 INIST/CNRS
File 434:SciSearch(R) Cited Ref Sci 1974-1989/Dec
         (c) 1998 Inst for Sci Info
File  34:SciSearch(R) Cited Ref Sci 1990-2004/Jul W2
         (c) 2004 Inst for Sci Info
File  99:Wilson Appl. Sci & Tech Abs 1983-2004/Jun
         (c) 2004 The HW Wilson Co.
File 583:Gale Group Globalbase(TM) 1986-2002/Dec 13
         (c) 2002 The Gale Group
File 266:FEDRIP 2004/May
         Comp & dist by NTIS, Intl Copyright All Rights Res
File  95:TEME-Technology & Management 1989-2004/Jun W1
         (c) 2004 FIZ TECHNIK
File 438:Library Lit. & Info. Science 1984-2004/Jun
         (c) 2004 The HW Wilson Co

Set     Items   Description
S1       6089   GARBAGE(2N)COLLECT? OR AUTOMAT?(2N)MEMOR???(2N)MANAG?
S2         14   (CALL()STACK? ? OR REGISTER? ?)(10N)HEAP
S3        213   (POINTER? ? OR IDENTIF???? OR IDENTIFICATION OR ADDRESS???
             OR MAP????)(7N)HEAP
S4         18   (POINTER? ? OR IDENTIF???? OR IDENTIFICATION OR ADDRESS???
             OR MAP????)(7N)(CALL()SITE? ?)
S5        231   DESCRIPTOR? ?(10N)(STACK()FRAME? ? OR REGISTER? ? OR TABLE?
              ? OR OFFSET? ? OR OFF()SET? ?)
S6       1521   (OFFSET? ? OR OFF()SET? ?)(7N)(POINTER? ? OR IDENTIF???? OR
              IDENTIFICATION OR ADDRESS??? OR MAP???? OR HEAP? ? OR STACK(-
             )FRAME? ?)
S7         55   CALL()STACK? ?
S8        211   CALL()SITE? ?
S9          0   FIRST()CALL()SITE? ?
S10     35321   DESCRIPTOR? ?
S11     10387   HEAP? ?
S12    142217   OFFSET? ? OR OFF()SET? ?
S13       158   STACK()FRAME? ?
S14         1   S1(30N)S2
S15        33   S1(30N)S3
S16         3   S1(30N)S4
S17         1   S1(30N)S5
S18         1   S1(30N)S6
S19         4   S1(30N)S7
S20         3   S1(30N)S8
S21         7   S1(30N)S10
S22       485   S1(30N)S11
S23        10   S1(30N)S12
S24        17   S1(30N)S13
S25        42   S14 OR S16:S21 OR S23:S24
S26        29   RD (unique items)
S27        24   S26 NOT PY=2002:2004
```

27/5/1 (Item 1 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

06298761 E.I. No: EIP03087364973
Title: Compact garbage collection tables
Author: Tarditi, David
Corporate Source: Microsoft Research, Redmond, WA 98052, United States
Conference Title: Proceedings of the International Symposium on Memory Management (ISMM 2000)
Conference Location: Minneapolis, MN, United States Conference Date: 20001015-20001016
Sponsor: ACM SIGPLAN
E.I. Conference No.: 60373
Source: Proceedings of the International Symposium on Memory Management 2000.
Publication Year: 2000
ISBN: 1581132638
Language: English
Document Type: CA; (Conference Article) Treatment: T; (Theoretical)
Journal Announcement: 0302W4

Abstract: **Garbage collection** tables for finding pointers on the stack can be represented in 20-25% of the space previously reported. Live **pointer** information is often the same at many **call sites** because there are few **pointers** live across most **call sites** . This allows live **pointer** information to be represented compactly by a small index into a table of descriptions of pointer locations. The mapping from program counter values to those small indexes can be represented compactly using several techniques. The techniques all assign numbers to call sites and use those numbers to index an array of small indexes. One technique is to represent an array of return addresses by using a two-level table with 16-bit off-sets. Another technique is to use a sparse array of return addresses and interpolate the exact number via disassembly of the executable code. 12 Refs.

Descriptors: *Storage allocation (computer); Program compilers; Codes (symbols); Encoding (symbols); Java programming language
Identifiers: Garbage collection tables
Classification Codes:
723.1.1 (Computer Programming Languages)
722.1 (Data Storage, Equipment & Techniques); 723.1 (Computer Programming); 723.2 (Data Processing)
722 (Computer Hardware); 723 (Computer Software, Data Handling & Applications)
72 (COMPUTERS & DATA PROCESSING)

27/5/2 (Item 2 from file: 8)
DIALOG(R)File 8:Ei Compendex(R)

06298536 E.I. No: EIP03087364742
Title: Efficient memory management in a merged heap/stack prolog machine
Author: Li, Xining
Corporate Source: Department of Computer Science Lakehead University, Thunder Bay, Ont., Canada
Conference Title: Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP'00)
Conference Location: Montreal, Que., Canada Conference Date: 20000920-20000923
Sponsor: ACM; SIGPLAN
E.I. Conference No.: 60363
Source: Proceedings of the 2nd International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming 2000.
Publication Year: 2000
ISBN: 1581132654
Language: English
Document Type: CA; (Conference Article) Treatment: T; (Theoretical)
Journal Announcement: 0302W4

Abstract: Traditional Prolog implementations are based on the stack/heap memory architecture: the stack holds local variables and control information, whereas the heap stores data objects which outlive procedure activations. A **stack frame** can be deallocated when an activation ends while heap space can only be reclaimed on backtracking or by **garbage collection** . Conventional **garbage collection** methods may yield poor performance. In this paper, I present a novel memory management approach used in the implementation of Logic Virtual Machine (LVM). The LVM combines the stack and the heap into a single memory block for all dynamical memory requirements, supports coarse-grain two-stream unification, and embeds an efficient garbage collection algorithm, the Chronological Garbage Collection (CGC), to reclaim useless memory cells. An experimental LVM emulator has been implemented. Our experimental results show that the proposed approach has low runtime overhead, good virtual memory and cache performance, and very short, evenly distributed pause times during garbage collection. Some benchmarks even revealed that the CGC not only improves the program's cache performance by more than enough to pay its own cost, but also improves the program execution performance which is competitive with the SICStus fast-code. 25 Refs.

Descriptors: *Data storage equipment; PROLOG (programming language); Cache memory; Costs; Algorithms

Identifiers: Memory management

Classification Codes:

723.1.1 (Computer Programming Languages)

722.1 (Data Storage, Equipment & Techniques); 723.1 (Computer Programming)

722 (Computer Hardware); 723 (Computer Software, Data Handling & Applications); 911 (Cost & Value Engineering; Industrial Economics); 921 (Applied Mathematics)

72 (COMPUTERS & DATA PROCESSING); 91 (ENGINEERING MANAGEMENT); 92 (ENGINEERING MATHEMATICS)

27/5/3 (Item 3 from file: 8)

DIALOG(R)File 8:Ei Compendex(R)
(c) 2004 Elsevier Eng. Info. Inc. All rts. reserv.

05832295 E.I. No: EIP01246536390

Title: Contaminated garbage collection

Author: Cannarozzi, D.J.; Plezbert, M.P.; Cytron, R.K.

Corporate Source: Washington University Department of Computer Science, St. Louis, MO 63130, United States

Conference Title: ACM SIGPLAN 2000 Conference on Programming Language Design and Implementation (PLDI)

Conference Location: Vancouver, BC, Canada Conference Date: 20000618-20000621

Sponsor: ACM SIGPLAN

E.I. Conference No.: 58101

Source: Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI) 2000. p 264-273

Publication Year: 2000

CODEN: PSPIEM

Language: English

Document Type: CA; (Conference Article) Treatment: T; (Theoretical)

Journal Announcement: 0106W3

Abstract: We describe a new method for determining when an object can be **garbage collected** . The method does not require marking live objects. Instead, each object X is dynamically associated with a **stack frame** M, such that X is collectable when M pops. Because X could have been dead earlier, our method is conservative. Our results demonstrate that the method nonetheless identifies a large percentage of collectable objects. The method has been implemented in Sun's Java trademark Virtual Machine interpreter, and results are presented based on this implementation. 20 Refs.

Descriptors: *Program debugging; Object oriented programming; Requirements engineering; Dynamic programming; Java programming language; Virtual reality; Program interpreters

Identifiers: Garbage collected languages; Java virtual machine interpreters

Classification Codes:

723.1.1 (Computer Programming Languages)

723.1 (Computer Programming); 921.5 (Optimization Techniques)

723 (Computer Software, Data Handling & Applications); 921 (Applied Mathematics)

72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS)

27/5/4 (Item 4 from file: 8)

DIALOG(R)File 8:Ei Compendex(R)

03983791 E.I. No: EIP94112421934

Title: Cut and side-effects in a data-driven implementation on Prolog

Author: Auwatanamongkol, Surapong; Ciepielewski, Andrzej; Biswas, Prasenjit

Corporate Source: Southern Methodist Univ, Dallas, TX, USA

Source: New Generation Computing v 12 n 3 1994. p 223-250

Publication Year: 1994

CODEN: NGCOE5 ISSN: 0288-3635

Language: English

Document Type: JA; (Journal Article) Treatment: G; (General Review)

Journal Announcement: 9501W1

Abstract: A number of data-driven execution models have been proposed for parallel execution of logic programs. LogDf is an abstract data-driven execution model for pure logic programs, which has shown promising performance during simulations. However, the original model lacks support for extra logical features such as cut and side-effects, which are needed to execute Prolog programs. This paper describes a scheme that has been incorporated into the LogDf model to support cut and side-effects. The main component of the scheme is a data structure, called a flat, non-strict S-Stream, which maintains strict ordering of multiple solutions, and, at the same time, allows simultaneous modification by several actors. This ordering corresponds to the order in which solutions would be produced in a sequential system and is necessary to implement cut and side-effects. The correct synchronization and ordering of operations on the cells of an S-Stream is ensured by the use of I-structure memory. The **descriptor** based token coloring mechanism in the LogDf provides convenient support for maintaining the scope information associated with cuts. An efficient **garbage collection** strategy is also proposed. (Author abstract) 15 Refs.

Descriptors: *Logic programming; Prolog (programming language); Data structures; Computer architecture; Parallel processing systems; Program assemblers; Sequential machines; Computational methods; Computer operating systems; Storage allocation (computer)

Identifiers: Execution model; Data driven execution; **Descriptor** based token coloring system; Efficient **garbage collection** strategy; Extra logical features

Classification Codes:

723.1.1 (Computer Programming Languages)

723.1 (Computer Programming); 723.2 (Data Processing); 721.3 (Computer Circuits); 721.1 (Computer Theory, Includes Formal Logic, Automata Theory, Switching Theory, Programming Theory); 921.6 (Numerical Methods)

723 (Computer Software); 721 (Computer Circuits & Logic Elements); 921 (Applied Mathematics)

72 (COMPUTERS & DATA PROCESSING); 92 (ENGINEERING MATHEMATICS)

27/5/5 (Item 5 from file: 8)

DIALOG(R)File 8:Ei Compendex(R)

02675761 E.I. Monthly No: EI8811104436

Title: GARBAGE COLLECTION OF STRINGS AND LINKED DATA STRUCTURES IN REAL TIME.

Author: Nilsen, K.

Corporate Source: Univ of Arizona, Tucson, AZ, USA

Source: Software - Practice and Experience v 18 n 7 Jul 1988 p 613-640

Publication Year: 1988
CODEN: SPEXBL   ISSN: 0038-0644
Language: English
Document Type: JA; (Journal Article)   Treatment: T; (Theoretical)
Journal Announcement: 8811

Abstract: This paper describes the addition of certain information to a string **descriptor** and enhancements to existing copying **garbage collection** algorithms that permit linked data structures and strings to be allocated and **garbage collected** from a shared region of memory in real time. This algorithm is real-time in the sense that the time required for allocation of each basic unit of memory is bounded by a constant. An analysis of performance is reported, and comparisons are made with traditional garbage collection. (Edited author abstract). 10 Refs.

Descriptors: *COMPUTER PROGRAMMING--*Algorithms; DATA PROCESSING--Data Structures; COMPUTER.SYSTEMS PROGRAMMING--Utility Programs
Identifiers: GARBAGE COLLECTION; SHARED MEMORY; REAL-TIME ALGORITHMS
Classification Codes:
723 (Computer Software)
72 (COMPUTERS & DATA PROCESSING)

27/5/6   (Item 1 from file: 35)
DIALOG(R)File 35:Dissertation Abs Online
(c) 2004 ProQuest Info&Learning. All rts. reserv.

01678366 ORDER NO: AAD99-11506
ISSUES IN THE DESIGN OF A JAVA PROCESSOR ARCHITECTURE (EMBEDDED PROCESSORS, WORLD WIDE WEB)
Author: NARAYANAN, VIJAYKRISHNAN
Degree: PH.D.
Year: 1998
Corporate Source/Institution: UNIVERSITY OF SOUTH FLORIDA (0206)
Major Professor: N. RANGANATHAN
Source: VOLUME 59/11-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
         PAGE 5939. 171 PAGES
Descriptors: COMPUTER SCIENCE ; ENGINEERING, ELECTRONICS AND ELECTRICAL
Descriptor Codes: 0984; 0544

With the popularization of the World Wide Web, the Java programming language has gained importance due to the growing need for the same executable code to run on different platforms. However, the execution speed of Java code is a major concern because of its interpreted nature. Some of Java's features that are desirable for building reusable and flexible software such as polymorphism, object manipulation, dynamic loading and resolution also contribute to slow execution. A Java processor architecture that implements the Java virtual machine (JVM) in hardware is proposed in this dissertation. This architecture enhances the execution speed by eliminating the interpretation overhead and also by providing customized support for Java execution. Various architectural support features are identified based on the study of Java execution characteristics using various benchmarks. Specifically, support for the stack processing, object manipulation and virtual method instructions that contribute to more than 70% of the instructions executed in these benchmarks are investigated.

Due to the stack based nature of the JVM, the operations to move data between the local variables of the stack frame and the operand stack constitute 50% of the executed instructions. An extended folding scheme to eliminate redundant data movement operations is proposed to address the problem. In this scheme, the execution of the data movement operations are combined with immediately preceding or following operations such as <italic>iadd</italic> that create or consume the moved data. The proposed scheme eliminates 2.6% of the instructions executed by a picoJava processor and 6% of the instructions executed by a pure stack machine. Fast object accessing and relocation capabilities are important for the efficient execution of object manipulation instructions and heap compactions that occur during **garbage collection** . Hence, an object cache **addressed** virtually by the (object reference, field **offset**) pair is proposed. This eliminates the additional indirection overhead associated with accessing objects with the handle representation in Sun's Java implementations. Also, it retains the capability to efficiently relocate objects unlike the direct access object models that can also be used to eliminate the indirection overhead. It is shown that on an average 1.5 cycles are reduced for each object access using the object cache when compared to the use of a handle model on conventional caches.

The frequent use of virtual method calls result in the execution of indirect branches in Java implementations that utilize a dispatch table. A path history based predictor is used to improve the prediction rate of the indirect branches as compared to the currently used branch target buffer (BTB) based schemes. Various parameters such as the number of history buffers, the path history length and the hashing scheme that influence the path history based predictor are customized based on Java code characteristics. It was observed that the misprediction rate for <italic>Javac</italic> benchmark reduces from 4.9% using an 8K BTB to 3.6% by using this method. Next, three hybrid cache schemes to implement virtual method calls instead of the currently used dispatch table techniques are studied. The hybrid cache schemes eliminate the time and space overhead associated with creating the dispatch table data structure. Instead, they exploit the receiver type locality and the low degree of polymorphism exhibited at the virtual methods call sites to provide an efficient virtual method invocation.

27/5/7 (Item 2 from file: 35)
DIALOG(R)File 35:Dissertation Abs Online

01473908 ORDER NO: AADAA-I9609899
R-CODE: A VERY CAPABLE VIRTUAL COMPUTER (LANGUAGE COMPILERS)
Author: WALTON, ROBERT LEE
Degree: PH.D.
Year: 1995
Corporate Source/Institution: HARVARD UNIVERSITY (0084)
Adviser: THOMAS E. CHEATHAM
Source: VOLUME 56/12-B OF DISSERTATION ABSTRACTS INTERNATIONAL.
PAGE 6877. 299 PAGES
Descriptors: COMPUTER SCIENCE
Descriptor Codes: 0984

This thesis investigates the design of a machine independent virtual computer, the R-CODE computer, for use as a target by high level language compilers. Unlike previous machine independent targets, R-CODE provides higher level capabilities, such as a **garbage collecting** memory manager, tagged data, type maps, array **descriptors** , **register** dataflow semantics, and a shared object memory. Emphasis is on trying to find universal versions of these high level features to promote interoperability of future programming languages and to suggest a migration path for future hardware.

The memory manager design combines both automatic garbage detection and an explicit "manual" delete operation. It permits objects to be copied at any time, to compact memory or expand objects. It traps obsolete addresses and instantly forwards copied objects using a software cache of an object map. It uses an optimized write-barrier, and is better suited for real-time than a standard copying collector.

R-CODE investigates the design of type maps that extend the virtual function tables of C++ and similar tables of HASKELL, EIFFEL, and SATHER 0.6. R-CODE proposes to include numeric types and sizes in type maps, and to inline information from type maps by using dynamic case statements, which switch on a type map identifier. When confronted with a type map not seen before, a dynamic case statement compiles a new case of itself to handle the new type.

R-CODE also investigates using IEEE floating point signaling-NaNs as tagged data, and making array descriptors first class data.

R-CODE uses a new "register dataflow" execution flow model to better match the coming generation of superscalar processors. Functional dataflow is used for operations on register values, and memory operations are treated as unordered I/O. Barriers are introduced to sequence groups of unordered memory operations. A detailed semantic execution flow model is presented.

R-CODE includes a shared object memory design to support multi-threaded programming within a building where network shared object memory reads and writes take several thousand instruction-execution-times to complete. The design runs on existing symmetric processors, but requires special caches to run on future within-building systems.

27/5/8 (Item 1 from file: 2)
DIALOG(R)File 2:INSPEC

6714854 INSPEC Abstract Number: C2000-11-7430-002
Title: Java Virtual Machines for resource-critical embedded systems and smart cards
Author(s): Golatowski, F.; Ploog, H.; Kraudelt, H.; Rachui, R.; Hagendorf, T.; Timmerman, O.D.
Author Affiliation: Rostock Univ., Germany
Conference Title: JIT'99. Java-Information-Tag 1999 p.121-34
Editor(s): Cap, C.H.
Publisher: Springer-Verlag, Berlin, Germany
Publication Date: 1999 Country of Publication: Germany xi+296 pp.
ISBN: 3 540 66464 5 Material Identity Number: XX-1999-02729
Conference Title: Proceedings of JIT'99: Java-Informations-Tage
Conference Date: 20-21 Sept. 1999 Conference Location: Dusseldorf, Germany
Medium: Also available on CD-ROM in PDF format
Language: German Document Type: Conference Paper (PA)
Treatment: Practical (P)
Abstract: Discusses the design and implementation of various embedded Java systems and illustrates Java Virtual Machines with Java loaders, linking and initialization units, **stack frames , garbage collectors** and an execution engine. Items in a Java class library are tabulated, and a smart-card Java processor is briefly described. Hardware access under Java is illustrated. (32 Refs)
Subfile: C
Descriptors: embedded systems; Java; smart cards; software libraries; storage management; virtual machines
Identifiers: Java Virtual Machines; resource-critical embedded systems; smart cards; Java loaders; linking units; initialization units; stack frames; garbage collectors; execution engine; Java class library; Java processor; hardware access
Class Codes: C7430 (Computer engineering); C6110J (Object-oriented programming); C6150N (Distributed systems software)
Copyright 2000, IEE

27/5/9 (Item 2 from file: 2)
DIALOG(R)File 2:INSPEC

6512166 INSPEC Abstract Number: C2000-04-6120-009
Title: The need for predictable garbage collection
Author(s): Reid, A.; McCorquodale, J.; Baker, J.; Hsieh, W.; Zachary, J.
Author Affiliation: Dept. of Comput. Sci., Utah Univ., Salt Lake City, UT, USA
Conference Title: WCSSS'99. ACM SIGPLAN Workshop on Compiler Support for System Software p.56-63
Publisher: Inst. Nat. Res. Inf. Autom, Le Chesnay, France
Publication Date: 1999 Country of Publication: France iii+102 pp.
Material Identity Number: XX-1999-01341
Conference Title: Proceedings of the 2nd Workshop on Compiler Support for System Software
Conference Date: 1 May 1999 Conference Location: Atlanta, GA, USA
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P)
Abstract: Modern programming languages such as Java are increasingly being used to write systems programs. By "systems programs" we mean programs that provide critical services (compilers), are long-running (Web

servers) or have time-critical aspects (databases or query engines). One of the requirements of such programs is predictable behavior. Unfortunately, predictability is often compromised by the presence of garbage collection. Various researchers have examined the feasibility of replacing garbage collection with forms of stack allocation that are more predictable than **garbage collection** , but the applicability of such research to systems programs has not been studied or measured. A particularly promising approach allocates objects in the nth **stack frame** (instead of just the top-most frame); we call this "deep stack allocation". We present dynamic profiling results for several Java programs to show that deep stack allocation should benefit systems programs, and we describe the approach that we are developing to perform deep stack allocation in Java. (19 Refs)

Subfile: C

Descriptors: Java; object-oriented programming; storage allocation; storage management; systems software

Identifiers: garbage collection; programming languages; systems programs; critical services; compilers; long-running programs; Web servers; time-critical aspects; databases; query engines; predictable behavior; deep stack allocation; stack frames; dynamic profiling; Java programs

Class Codes: C6120 (File organisation); C6150 (Systems software); C6110J (Object-oriented programming)

Copyright 2000, IEE

27/5/10 (Item 3 from file: 2)

DIALOG(R)File 2:INSPEC
(c) 2004 Institution of Electrical Engineers. All rts. reserv.

6450277 INSPEC Abstract Number: C2000-02-6120-011

Title: Exploring single stack architecture for Prolog

Author(s): Xining Li

Author Affiliation: Dept. of Comput. Sci., Lakehead Univ., Thunder Bay, Ont., Canada

Conference Title: Proceedings of the Seventeenth IASTED International Conference. Applied Informatics p.489-91

Editor(s): Hamza, M.H.

Publisher: ACTA Press, Anaheim, CA, USA

Publication Date: 1999 Country of Publication: USA 699 pp.

ISBN: 0 88986 241 9 Material Identity Number: XX-1999-00795

Conference Title: Proceedings of 17th IASTED International Conference on Applied Informatics (AI'99)

Conference Sponsor: IASTED

Conference Date: 15-18 Feb. 1999 Conference Location: Innsbruck, Austria

Language: English Document Type: Conference Paper (PA)

Treatment: Practical (P)

Abstract: Traditional Prolog implementations are based on stack/heap memory architecture: stack holds temporary variables and control information, whereas the heap stores dynamical data objects. **Stack frames** can be deallocated on the return of procedure calls while heap space can only be reclaimed on backtracking or by **garbage collection** . Conventional GC algorithms may yield poor performance. The reason for using stack/heap architecture is that deallocating **stack frames** is in fact cheap, incremental **garbage collection** . I present a novel memory management approach used in the implementation of Logic Virtual Machine (LVM). Different from the well-known Warren's Abstract Machine which uses the structure copying method, the LVM adopts a hybrid of program sharing and structure copying to represent Prolog terms. A new point of LVM is that it explores a single stack paradigm for all dynamical memory requirements and embeds an efficient garbage collection algorithm, Chronological Garbage Collection (CGC), to reclaim useless memory cells. Our early results show that the proposed approach has low runtime overhead, good virtual memory and cache performance, and very short, evenly distributed pause times. Some benchmarks even revealed that the CGC improves the program's cache performance by more than enough to pay its own cost. (8 Refs)

Subfile: C

Descriptors: memory architecture; PROLOG; storage management

Identifiers: single stack architecture; Prolog; memory management

approach; Logic Virtual Machine; program sharing; structure copying; dynamical memory requirements; Chronological Garbage Collection; runtime overhead; virtual memory performance; cache performance; pause times; benchmarks; stack memory architecture; heap memory architecture
Class Codes: C6120 (File organisation)
Copyright 1999, IEE

27/5/11 (Item 4 from file: 2)
DIALOG(R)File 2:INSPEC
(c) 2004 Institution of Electrical Engineers. All rts. reserv.

5513171 INSPEC Abstract Number: C9704-6120-016
Title: Abstract models of memory management
Author(s): Morrisett, G.; Felleisen, M.; Harper, R.
Author Affiliation: Carnegie Mellon Univ., Pittsburgh, PA, USA
Conference Title: Conference Record of FPCA '95. SIGPLAN-SIGARCH-WG2.8. Conference on Functional Programming Languages and Computer Architecture p.66-77
Publisher: ACM, New York, NY, USA
Publication Date: 1995 Country of Publication: USA viii+333 pp.
ISBN: 0 89791 719 7 Material Identity Number: XX95-01353
U.S. Copyright Clearance Center Code: 0 89791 719 7/95/0006.$3.50
Conference Title: Proceedings of 7th Annual SIGPLAN/SIGARCH/WG2.8 Conference on Functional Programming Languages and Computer Architecture
Conference Sponsor: ACM SIGPLAN; ACM SIGARCH; IFIP
Conference Date: 25-28 June 1995 Conference Location: La Jolla, CA, USA
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P); Theoretical (T)
Abstract: Most specifications of **garbage collectors** concentrate on the low-level algorithmic details of how to find and preserve accessible objects. Often, they focus on bit-level manipulations such as scanning **stack frames** , marking objects, tagging data, etc. While these details are important in some contexts, they often obscure the more fundamental aspects of memory management: what objects are garbage and why? We develop a series of calculi that are just low-level enough that we can express allocation and garbage collection, yet are sufficiently abstract that we may formally prove the correctness of various memory management strategies. By making the heap of a program syntactically apparent, we can specify memory actions as rewriting rules that allocate values on the heap and automatically dereference pointers to such objects when needed. This formulation permits the specification of garbage collection as a relation that removes portions of the heap without affecting the outcome of the evaluation. Our high-level approach allows us to specify in a compact manner a wide variety of memory management techniques, including standard trace-based garbage collection (i.e., the family of copying and mark/sweep collection algorithms), generational collection, and type-based, tag-free collection. Furthermore, since the definition of garbage is based on the semantics of the underlying language instead of the conservative approximation of inaccessibility, we are able to specify and prove the idea that type inference can be used to collect some objects that are accessible but never used. (30 Refs)
Subfile: C
Descriptors: formal specification; functional programming; process algebra; storage allocation; storage management; type theory
Identifiers: memory management abstract models; specifications; garbage collectors; bit-level manipulations; programming calculi; storage allocation; correctness proof; heap; rewriting rules; pointers; high-level approach; trace-based garbage collection; generational collection; type-based tag-free collection; type inference; functional programming
Class Codes: C6120 (File organisation); C4240 (Programming and algorithm theory); C6110 (Systems analysis and programming)
Copyright 1997, IEE

27/5/13 (Item 6 from file: 2)
DIALOG(R)File 2:INSPEC

4509121 INSPEC Abstract Number: C9312-6150C-027
Title: FCG: a code generator for lazy functional languages
Author(s): Langendoen, K.; Hartel, P.H.
Author Affiliation: Amsterdam Univ., Netherlands
Conference Title: Compiler Construction. 4th International Conference CC'92 Proceedings p.278-96
Editor(s): Kastens, U.; Pfahler, P.
Publisher: Springer-Verlag, Berlin, Germany
Publication Date: 1992 Country of Publication: West Germany viii+320 pp.
ISBN: 3 540 55984 1
Conference Date: 5-7 Oct. 1992 Conference Location: Paderborn, Germany
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P)

Abstract: The FCG code generator produces portable code that supports efficient two-space copying **garbage collection** . The code generator transforms the output of the FAST compiler front end into an abstract machine code. This code explicitly uses a **call stack** , which is accessible to the **garbage collector** . In contrast to other functional language compilers that generate assembly directly, FCG uses the C compiler for code generation, providing high-quality code optimisations and portability. To make full use of the C compiler's capabilities, FCG includes an optimisation scheme that transforms the naively generated stack-based code into a register-based equivalent form. The results for a benchmark of functional programs show that code generated by FCG performs well in comparison with the LML compiler. (14 Refs)

Subfile: C
Descriptors: C language; functional programming; program compilers; storage management
Identifiers: FCG; code generator; lazy functional languages; portable code; two-space copying garbage collection; FAST compiler front end; abstract machine code; call stack; functional language compilers; C compiler; code optimisations; LML compiler
Class Codes: C6150C (Compilers, interpreters and other processors); C6140D (High level languages)

27/5/14 (Item 7 from file: 2)
DIALOG(R)File 2:INSPEC

04131835 INSPEC Abstract Number: C9205-4240-026
Title: CONS should not CONS its arguments, or, a lazy alloc is a smart alloc (high level languages)
Author(s): Baker, H.G.
Author Affiliation: Nimble Comput. Corp., Encino, CA, USA
Journal: SIGPLAN Notices vol.27, no.3 p.24-34
Publication Date: March 1992 Country of Publication: USA
CODEN: SINODQ ISSN: 0362-1340
Language: English Document Type: Journal Paper (JP)
Treatment: Bibliography (B); Practical (P); Theoretical (T)

Abstract: Discusses lazy allocation, a model for allocating objects on the execution stack of a high-level language which does not create dangling references. The author's model provides safe transportation into the heap for objects that may survive the deallocation of the surrounding stack frame. Space for objects that do not survive the deallocation of the surrounding **stack frame** is reclaimed without additional effort when the stack is popped. Lazy allocation thus performs a first-level **garbage collection** , and if the language supports **garbage collection** of the heap, then the model can reduce the amortized cost of allocation in such a heap by filtering out the short-lived objects that can be more efficiently managed in LIFO order. Important applications of the model include the efficient allocation of temporary data structures that are passed as arguments to anonymous procedures which may or may not use these data structures in a stack-like fashion. (74 Refs)

Subfile: C

Descriptors: data structures; functional programming; high level languages; programming theory
Identifiers: object allocation; optimization; functional programming; lazy allocation; execution stack; high-level language; stack frame; garbage collection; data structures
Class Codes: C4240 (Programming and algorithm theory); C6110 (Systems analysis and programming)

27/5/15 (Item 8 from file: 2)
DIALOG(R)File 2:INSPEC

02605741 INSPEC Abstract Number: C86014082
Title: PROLOG execution in Simula
Author(s): Lamy, J.-F.; Vaucher, J.
Author Affiliation: Dept. d'inf. et de Recherche Oper., Montreal Univ., Ont., Canada
Conference Title: Proceedings of the Thirteenth SIMULA Users' Conference p.61-71
Publisher: SIMULA, Oslo, Norway
Publication Date: 1985 Country of Publication: Norway ii+133 pp.
Conference Date: 28-30 Aug. 1985 Conference Location: Calgary, Alta., Canada
Language: English Document Type: Conference Paper (PA)
Treatment: Practical (P)
Abstract: Object-oriented design has allowed the authors to build an extremely modular inference engine for PROLOG, a nonprocedural language based on logic. The prototype includes most techniques used in prominent PROLOG implementations, but expressed with high-level language constructs. The authors make extensive use of the class/sub-class concepts to emulate compilation. The Search of the solution space is expressed elegantly via coroutines. The resident **garbage collector** manages PROLOG variable bindings and reclaims **stack frames** automatically when terminal recursivity occurs. The resulting interpreter is a good vehicle for experimentation of intelligent search strategies. (9 Refs)
Subfile: C
Descriptors: logic programming; PROLOG; simulation languages
Identifiers: compiler emulation; PROLOG execution; Simula; modular inference engine; nonprocedural language; PROLOG implementations; high-level language constructs; garbage collector; reclaims stack frames; terminal recursivity; intelligent search strategies
Class Codes: C6110 (Systems analysis and programming); C6140D (High level languages); C6150 (Systems software)

27/5/24 (Item 1 from file: 99)
DIALOG(R)File 99:Wilson Appl. Sci & Tech Abs

1277006 H.W. WILSON RECORD NUMBER: BAST95072652
A jolt of Java could shake up the computing community
Carlson, Bob;
Computer v. 28 (Nov. '95) p. 81-2
DOCUMENT TYPE: Feature Article ISSN: 0018-9162 LANGUAGE: English
RECORD STATUS: New record

ABSTRACT: Sun Microsystems has developed a network programming language called Java that is set to accelerate the trend of viewing the PC as merely playing a supporting role to the Internet. Java is an object-oriented language that resembles C++, but it is hardware and implementation independent. It creates a virtual machine on the client computer that comprises the following logical abstract components: an instruction set, a set of **registers** , a stack, a **garbage - collected heap** , and a method area. The byte-code instructions of Java programs are translated by the Java interpreter into virtual machine instructions that can be executed on any machine to which the interpreter has been ported.